1 Graph Data Structures

Recall that a graph $G = (V, E)$ is a pair of sets, where V is the set of vertices and E is the set of edges.

Question: How should we represent a graph?
There are two standard ways of representing graphs: The adjacency matrix and the adjacency list.
For this section, lets let $n = |V|$.

Adjacency Matrix Let A be a 2-dimensional $n \times n$ array, and let

$$A[i][j] = \begin{cases} 1 & \text{if there is an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

Adjacency List Let A be a 1-dimensional array of n linked lists. Then the list $A[i]$ contains vertices adjacent to i.

Each of these representations has advantages and disadvantages, depending on the problem you are trying to solve and the properties of the graph.
For example:

Adjacency Matrix

- $\Theta(|V|^2)$ space required
- Lookup for a specific edge takes constant time
- There are useful matrix operations

Adjacency List

- $\Theta(|V| + |E|)$ space required
- Lookup the neighbors of a vertex faster

In practice, most graphs are "sparse" (not "dense"), meaning is has few edges. For example, Facebook has over a billion users, but on average each user has a few friends, maybe ≤ 1000.
In a graph like this, much less space is required for the adjacency list than for the adjacency matrix.
Lets play around with the adjacency matrix a little bit

Calculate A^2
Question: Is A^2 interesting?
Question: Is A^3 interesting?
Question: Let k be any positive integer, is A^k interesting?

See video for the answers to these questions.

One useful thing we can do with the adjacency matrix is compute the transitive closure of the graph.

For a directed graph $G = (V, E)$, the **transitive closure of** G is a directed graph $G^* = (V, E^*)$ where

$$E^* = \{(v, w) : v, w \in V \text{ and there is a path in } G \text{ from } v \text{ to } w\}.$$

If A is the adjacency matrix of G and A^* is the adjacency matrix of G^*, we can compute A^* from A by making the following observations:

$A^0 = I$ tell us the paths of length 0,

A tells us the paths of length 1,

A^2 tells us the paths of length 2,

A^3 tells us the paths of length 3, and so on.

Note that we only need to compute up to the $(n-1)$-th power, since any path of length greater than $n-1$ would have to contain a repeated vertex, and thus there would be a shorter path with the same endpoints computed by one of the earlier powers.
Using this, do we have that

$$A^* = I + A + A^2 + A^3 + \ldots + A^{n-1}?$$

Not exactly, some entries in the matrix could have values > 1.
Instead, we have that

$$A^* = g(I + A + A^2 + A^3 + \ldots + A^{n-1})$$

where g is a function that sets all values > 1 to 1.
This leads us to the following algorithm:

Algorithm1(A)
$M = I$
for $i = 1$ to $n - 1$ do
    $M = I + MA$
return $g(M)$

This algorithm has running time $\Theta(nT(n))$, where $T(n)$ is the running time of the matrix multiplication algorithm we use.
We can speed this algorithm up in the following way:

Algorithm2(A)
$M = I + A$
for $i = 1$ to $\lceil \log_2(n - 1) \rceil$ do
    $M = M^2$
return $g(M)$

This algorithm has running time $\Theta(\log(n)T(n))$, where $T(n)$ is the running time of the matrix multiplication algorithm we use, a significant improvement.